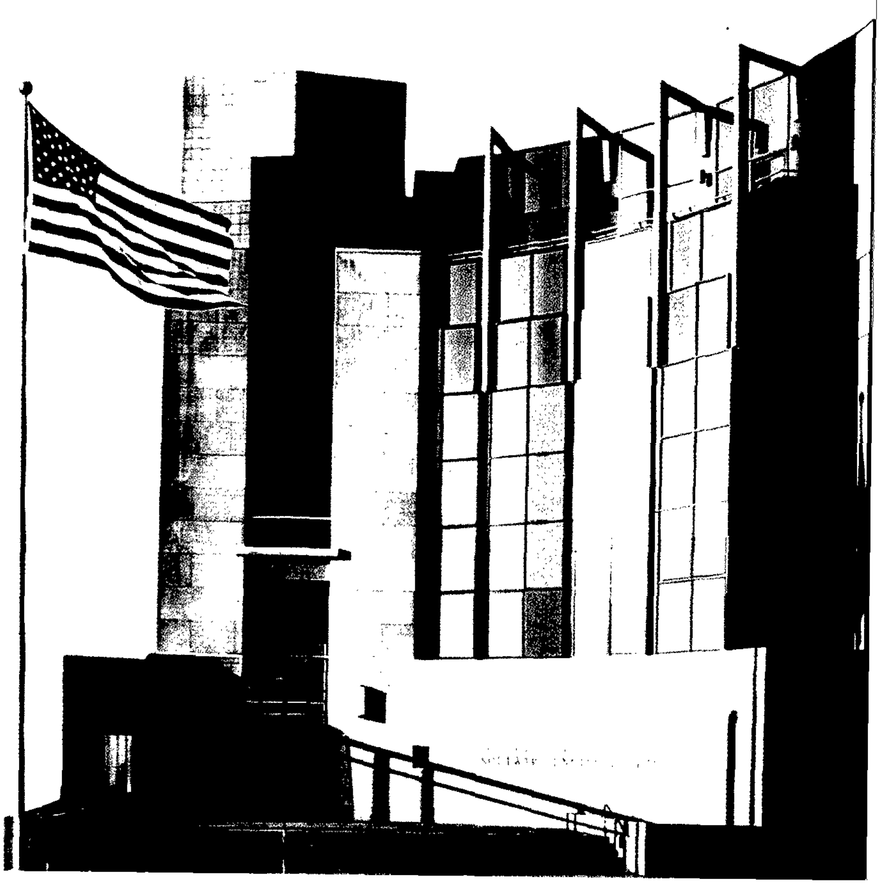
Carnegie Mellon
Software Engineering Institute

Guidelines for Software Engineering Education Version 1.0

Donald J. Bagert, Texas Tech University
Thomas B. Hilburn, Embry-Riddle Aeronautical University
Greg Hislop, Drexel University
Michael Lutz, Rochester Institute of Technology
Michael McCracken, Georgia Institute of Technology
Susan Mengel, Texas Tech University

November 1999

19991109 159



TECHNICAL REPORT
CMU/SEI-99-TR-032
ESC-TR-99-002

Carnegie Mellon
Software Engineering Institute
Pittsburgh, PA 15213-3890

Guidelines for Software Engineering Education Version 1.0

CMU/SEI-99-TR-032
ESC-TR-99-002

Donald J. Bagert, Texas Tech University
Thomas B. Hilburn, Embry-Riddle Aeronautical University
Greg Hislop, Drexel University
Michael Lutz, Rochester Institute of Technology
Michael McCracken, Georgia Institute of Technology
Susan Mengel, Texas Tech University

November 1999

Networked Systems Survivability Program

The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Table of Contents

List of Figures

List of Tables

Preface

The last twenty years have witnessed significant advancements in the state of computer science education (and in allied fields such as computer engineering and information systems). The Association for Computing Machinery (ACM), the IEEE Computer Society (IEEE-CS), and the Computer Sciences Accreditation Board (CSAB) have provided encouragement, support, and guidance in developing quality curricula that are viable and dynamic. We have moved from language and coding centered curricula to programs that emphasize theory, abstraction, and design. However, most academic programs in computing still devote little time to areas such as requirements modeling, design methods, architecture, reuse, software processes, quality issues, team skills, and other areas of software engineering essential to effective commercial software development. Members of the Working Group on Software Engineering Education and Training (WGSEET) have prepared this document to provide guidance and support for the development of undergraduate computing curricula that better support the education of software engineering professionals.

In recent years, there has been a burst of interest and activities associated with advancing the state of the software engineering profession. Since 1993, the IEEE-CS and ACM have been actively promoting software engineering as a profession. Special task forces have been directed at establishing a software engineering code of ethics and professional practice, determining accreditation criteria for software engineering programs supporting curriculum development, and preparing a guide to the software engineering body of knowledge. In 1998, the IEEE-CS and the ACM formed the Software Engineering Coordinating Committee (SWECC) to foster the evolution of software engineering as a professional computing discipline and to coordinate the various software engineering "maturing" activities. An early draft of the *Guidelines*, that included a software engineering body of knowledge, was given to the SWECC project that is developing a guide to the software engineering body of knowledge. Another SWECC project is the Software Engineering Education Project (SWEEP). It was established to develop curriculum recommendations for software engineering programs. The WGSEET has shared its work on the *Guidelines* with SWEEP and supports the objectives of SWEEP. As the SWEEP efforts progress, WGSEET will continue to share its work and pursue complementary projects related to the improvement of software engineering education.

The two central parts of the *Guidelines* are the description of a software engineering body of knowledge and a curriculum model. The body of knowledge presents a high-level organization and description of the software engineering that supports effective curriculum design; and the curriculum model consists of a design architecture, a set of design concepts, and curriculum content guidance. We believe that this material and other guidance offered will provide assistance to faculty in the design and development of quality programs in software engineering and related curricula.

1 Introduction

Into the foreseeable future, software will play an increasingly important and central role in all aspects of daily life. The number, size, complexity, and application domains of programs being developed will grow dramatically. Unfortunately, there are serious problems in the cost, timeliness, and quality of development of many software products. The code in consumer products is doubling every two years; it is almost the norm for software projects to overrun their planned cost and schedule. Many large-scale development projects are never completed, and many of those completed do not meet the user requirements and are of poor quality [Gibbs 94]. These issues have placed an increasing demand for software developers who are equipped not only to deal with the scientific and technical aspects of computing, but for those who have professional education and preparation for the practice of software engineering [Denning 92, Ford 96, Lethbridge 98]. This includes practice related to use of software processes, measurement and analysis, front-end development methods, quality engineering, software maintenance, testing, and working as part of a team. Unfortunately, too few academic programs offer curricula that focus on this type of education. The solution to this problem depends heavily on the ability of faculty to redesign and implement curricula that not only emphasize computer science, information science, and technology, but also focus on the "practice" of software engineering (SE) and include the equally critical people and process issues. The main purpose of these Guidelines is to provide assistance to faculty in the design and development of programs in software engineering and related curricula.

2 Objectives

The specific objectives of these Guidelines are as follows:

- Promote discussion among professionals in education and industry that will improve software education in all institutions at an international level.
- Encourage greater commonality in software education within and across computing disciplines.
- Provide a coherent, structured description of software engineering concepts, knowledge, and practices that support software education curriculum development.
- Develop a model curriculum for software engineering that can be applied in whole, or part, to the development of software education programs.

3 Guidelines Development

In November 1997, the Working Group on Software Engineering Education and Training (WGSEET) met in Pittsburgh and discussed the need for a set of guidelines to support the design and implementation of software engineering courses and curricula. The WGSEET is a group of industry and academic professionals interested in promoting the development and future outlook of software engineering education and training [WGSEET]. The WGSEET meets twice a year (in fall at the Frontiers in Education Conference, and in spring at the Conference on Software Engineering Education and Training) to engage in activities that promote and support the education and training of software engineers. Although there are existing curriculum guides produced for various computing curricula [Barnes 98, CSAB 98, Tucker 91] and some early work done to support software engineering curriculum development [BCS 89, Ford 90], there is no existing document that provides broad and comprehensive information and direction for the development of undergraduate programs in software engineering. In the past two years one of the primary activities of the WGSEET has been to address this deficiency by preparing this document.

4 Software Engineering Body of Knowledge

Software engineering as a field of study and practice is a relatively new discipline. Some would argue that it is not even an engineering discipline; however, a more prevalent view is that it is simply evolving and not yet mature [Ford 94]. However, in order to provide curriculum guidance in software engineering education there is a need for a description and understanding about what knowledge makes up the field of software engineering. There have been recent efforts to organize and define a software engineering body of knowledge. The IEEE-CS and the ACM have a long-term effort underway to develop a comprehensive body of knowledge for software engineering [Dupuis 98]. A body of knowledge was developed to help define and assess the software competencies needed in a software-intensive organization [Hilburn 99]. However, for the purpose of this project, we felt that a simpler and more focused description of software engineering knowledge was needed. In this section, we present a high-level organization and description of the software engineering body of knowledge (SE-BOK) that supports effective curriculum design. (Note that SWEBOK is the abbreviation used in *A Guide to the Software Engineering Body of Knowledge* [Dupuis 98] to designate the software engineering body of knowledge.) The SE-BOK is organized into four "knowledge areas:" the Core Area, the Foundations Area, the Recurring Area, and the Supporting Area. Each knowledge area is further subdivided into "knowledge components." Definitions for the various knowledge terms used in this document are defined in Table 1. In the following sections, we describe each knowledge area and its components.

Term	Definition
Knowledge	the whole spectrum of content for the discipline: information, terminology, artifacts, data, roles, methods, models, procedures, techniques, practices, processes, and literature
Body of knowledge (BOK)	a hierarchical description of software engineering knowledge that organizes and structures the knowledge into two levels of hierarchy: "knowledge areas" and "knowledge components"
Knowledge Area (KA)	a sub-discipline of software engineering that represents a significant part of the body of software engineering knowledge. Knowledge areas are high-level structural elements used for organizing, classifying, and describing software engineering knowledge. A knowledge area is composed of a set of "knowledge components."
Knowledge Component (KC)	a subdivision of a KA that represents SE knowledge that is logically cohesive and related to the KA through inheritance or aggregation

Table 1: Knowledge Definitions

4.1 Core Area

The Core Area includes those components that define the essence of software engineering:

- Software Requirements
- Software Design
- Software Construction
- Software Project Management
- Software Evolution

A. Software Requirements Component

The Software Requirements component includes knowledge concerned with establishing a common understanding of the requirements to be addressed by a software project. It includes methods, techniques, and tools associated with the collection, analysis, specification, and review of software requirements.

B. Software Design Component

The Software Design component includes knowledge about principles, methods and techniques for describing how a software product will be implemented so that its requirements are satisfied. This includes methods, techniques, and tools associated with the various types of design activities: architectural design, component design, interface design, data design, and algorithm design.

C. Software Construction Component

The Software Construction component includes, in addition to knowledge about language syntax, issues related to coding style and standards, internal documentation, code prototyping, code reuse, analysis and choice of implementation tools, and implementation strategies in the use of various language paradigms (such as assembler, procedural, and object-oriented).

D. Software Project Management Component

The Software Project Management component addresses issues involving the creation, development, and maintenance of software projects. This area includes project management, risk analysis, project planning, project administration, and configuration management.

E. Software Evolution Component

The Software Evolution component addresses the knowledge and techniques necessary to enhance, perfect, and modify software over time. It includes the issues of software maintenance, extensibility, software adaptability to different environments, and software reengineering.

4.2 Foundations Area

The Foundations Area includes those components which undergird the Core Area and Recurring Area (explained on page 10). The Foundations Area consists of the following components:

- Computing Fundamentals
- Human Factors
- Application Domains

A. Computing Fundamentals Component

The Computing Fundamentals component addresses those topics upon which effective software development can be built. The topics include foundations of computing, programming, programming language concepts, history of computing, data structures, algorithm analysis, compiler organization, computer complexity, operating systems, and computer architecture.

B. Human Factors Component

The Human Factors component encompasses the two broad categories characterized by users who utilize software and by system developers who construct software. In these two categories, interface design, ergonomics, development environments, team dynamics, and communication are of primary concern to facilitate and make efficient the human-computer interface.

C. Application Domains Component

The Application Domains component includes knowledge about how software engineering techniques should be used effectively in various application domains: real-time systems, database and information systems, high-performance computing, intelligent systems, graphics applications, systems software, and so on.

4.3 Recurring Area

Components in the Recurring Area are threads that occur through all of the Core Area components. The Recurring components include the following:

- Ethics and Professionalism
- Software Processes
- Software Quality
- Software Modeling
- Software Metrics
- Tools and Environments
- Documentation

A. Ethics and Professionalism Component

The Ethics and Professionalism component addresses those ethical, social, and professional issues inherent in producing software and interacting with colleagues and clients. This component includes legal, quality, confidentiality, environmental, safety, workplace and harassment, and professional certification concerns.

B. Software Processes Component

The Software Processes component involves the definition, implementation, evaluation, and improvement of software engineering processes. It includes models for categorizing software maturity and applying software skills and techniques.

C. Software Quality Component

The Software Quality component encompasses the techniques and skills necessary to assure that software meets its requirements, that software development and maintenance processes are sound and measurable, and that, when required, software exhibits the characteristics of survivability, safety, and fault tolerance.

D. Software Modeling Component

The Software Modeling component covers principles and methods for modeling software architectures and software development entities. This includes techniques for using abstraction, modularity and hierarchy to model software functionality, data object relationships, behavior models, and formal methods.

E. Software Metrics Component

The Software Metrics component encompasses the tools and techniques for performing experimental analysis on, keeping historical records on, and monitoring the progress of software engineering processes, artifacts, and resources. It involves metrics definitions, data collection, and experimental techniques.

F. Tools and Environments Component

The Tools and Environments component includes knowledge about the tools and environments that automate software processes at various stages of the software life cycle. It includes tool analysis, selection, and operation.

G. Documentation Component

The Documentation component encompasses all of the material required to support the building and later maintenance of a software product. It includes methods, tools, and standards to write clear and accessible material for all artifacts and processes of the software life cycle.

4.4 Supporting Area

The components in the Supporting Area include other fields of study which provide building blocks for rounding out the education of students in software engineering. They include, but are not limited to, general education, mathematics, natural sciences, social sciences, business studies, and engineering.

5 A Software Engineering Curriculum Model

The curriculum model shown in the figure below is meant to provide a design abstraction that could be used to develop a variety of software engineering curricula. The model's development was influenced by a number of sources and experiences [Cowling 98, Hilburn 97, Naveda 98]. It consists of a design architecture, a set of design concepts, and curriculum content guidance. Although the actual implementation of a curriculum design depends on a number of requirements and constraints posited by an academic institution and its faculty, we believe this model can provide insight and structure that will help in developing a quality curriculum.

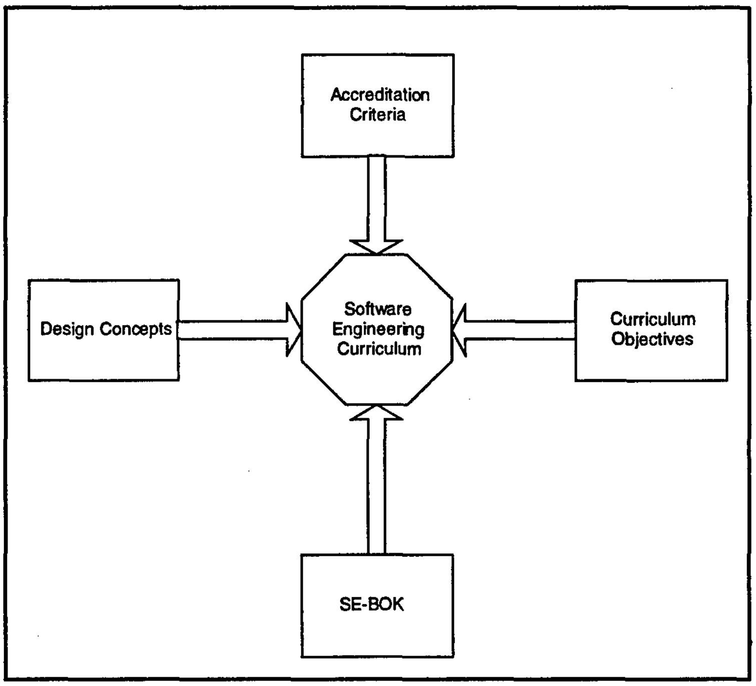


Figure 1: Key Elements of Curriculum Design

5.1 A Profile of Graduates

A degree program focused on software engineering is designed to prepare a new generation of software developers who concentrate on the engineering of software systems that meet specified requirements, are built with industrial quality standards, and are within cost and schedule. Graduates of such programs will be able to function as proficient software developers and effective team members. They will have preparation in communication, mathematics and science, and the cultural, historical and social issues that influence and affect software development. They will have knowledge about and experience with software product engineering and engineering management and an understanding of professional issues and practice. Graduates will be able to understand and assess their own software capabilities and performance. They will also have the knowledge and ability to assess the competencies of others working on a project team. This will allow them, with appropriate practice and experience, to assume leadership positions in a software development organization [Diaz-Herrera 99].

5.2 Curriculum Architecture

When developing a curriculum, there are key elements that must be considered in its design. The first and perhaps most important step is to determine the curriculum's objectives. This involves consideration of the institution's mission and the strategic goals of the department. At a more fundamental level, all the stakeholders (faculty, students, employers, etc.) in the curriculum need to be involved in setting these objectives.

The SE-BOK provides the foundation for the curriculum content, while accreditation criteria gives structure and discipline to the curriculum design and forms the basis for assessment and evaluation. But objectives, assessment criteria, and content are not alone sufficient to design a quality curriculum. There must be an underlying philosophy of how these elements fit together. The design concepts presented in the next section furnish a philosophical framework for the development of effective software engineering curricula and are the foundation for our curriculum model.

5.3 Design Concepts

The list of concepts in this section provides both foundation principles for the software engineering curriculum model and specific suggestions for creating a design.

A. The curriculum model supports the development of a variety of degree programs which emphasize and focus on software engineering:

   - BS in Software Engineering
   - BS in Computer Science
   - BS in Information Systems
   - BS in Computer Engineering

B. The model introduces students, at the beginning of their studies, to the nature, scope, and importance of software engineering.

C. The model incorporates two levels of software engineering education:

   i) "Software Engineering in the Small"

      ➢ the application of software engineering principles to the development of a software product by an individual

      ➢ the software developed is typical of that encountered in lower division software courses

   ii) "Software Engineering in the Large"

      ➢ the application of software engineering principles to the development of a software product by a team

      ➢ the software developed is typical of that encountered in upper division software courses, or small to moderate size industrial software projects

D. The model advocates a balance between "product" activities and "process" activities.

   - Product activities include methods, techniques, and skills used to build the components and artifacts associated with a software product (development plans, quality assurance plans, requirements specifications, design specifications, code, and product documentation).
   - Process activities include the standards, procedures, guidelines, measurement, and support and analysis techniques that provide a framework for how to carry out the product activities in an effective and efficient manner.

E. The model provides guidance for development of degree programs that can be accredited by an appropriate accreditation organization such as the Accreditation Board for Engi-

neering and Technology (ABET) or the Computing Sciences Accreditation Board (CSAB).

F. The model includes ethical, professional, and social issues that are related to the practice of software engineering.

G. The model places special emphasis on the importance of teamwork, and the team dynamics and team-building efforts and experiences that support effective collaborative software development activities.

H. The model provides for acquisition of knowledge in a particular application domain (e.g., embedded systems, database and information systems, intelligent systems, telecommunications and networking, etc.).

I. A key feature of the model is the provision for students to engage in the practice of software engineering. Such "practice" can be realized in a variety of ways:

   i) scheduled laboratories offered in conjunction with specific computer science and software engineering knowledge courses

   ii) software engineering project labs

      (1) Software Studio model (Carnegie Mellon University, [Garlan 97])

      (2) Real World Lab model (Georgia Institute of Technology, [Moore 94])

   iii) cooperative education and intern work in the software industry

5.4 Curriculum Content

Although the organization, implementation, and delivery of various SE-focused curricula might differ, they should all offer courses and activities that include the content areas depicted in Figure 2. The "domain knowledge" area is recognition that software engineering is applied in a variety of domains that each require specialized knowledge by the software developer. For example, the development of a financial software package requires quite different knowledge and skills than the development of software for an embedded flight control system. Of course the domain area of a curriculum also influences some of the decisions on what to include in the other content areas (e.g., the type and level of courses needed in mathematics, the natural sciences, and computer science).

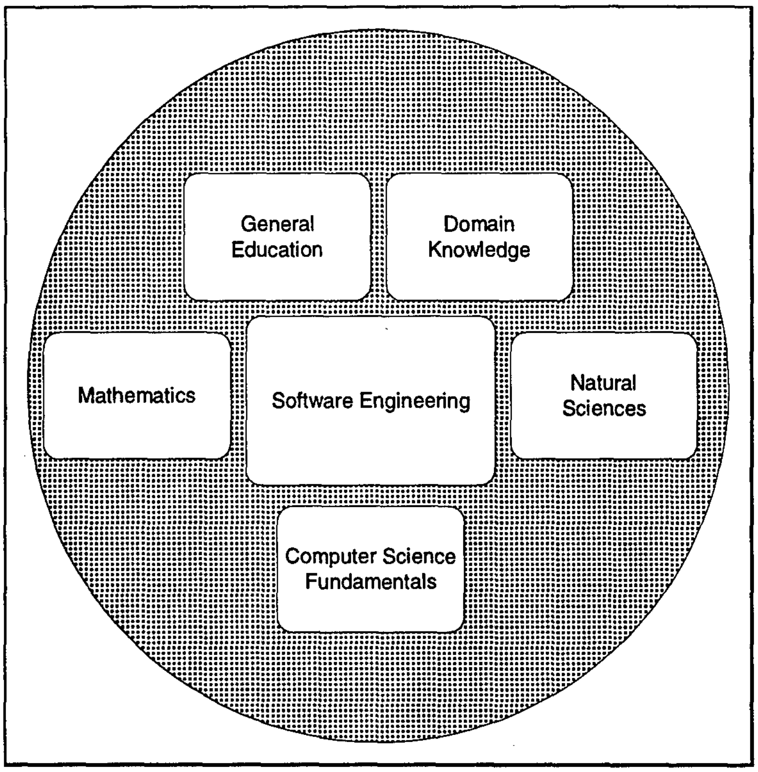


Figure 2: Curriculum Content Areas

The following list describes, as a minimum, what we believe should be included in each area:

A. Computer Science Fundamentals

- programming, data structures, and algorithms
- programming language concepts
- computer organization
- system software (process and resource management, concurrent and distributing computing, networking, and telecommunications)

B. Mathematics

- discrete math
- probability and statistics

C. Natural Sciences

D. General Education

- Communication (oral and written)
- Humanities and Social Sciences

E. Software Engineering

- Software Requirements Engineering
- Software Design
- Software Quality
- Software Architectures
- Software Construction
- Software Evolution
- Formal Methods
- HCI (Human-Computer Interaction)
- project organization, planning, and tracking
- Software Processes
- Software Engineering Ethics and Professionalism

5.5 Software Engineering Modules

This section includes a specification for software engineering modules that would form the core for any computing program with a focus in software engineering. The modules are based on the curriculum content listed above and represent material that should be incorporated in required courses for the curriculum. Table 2 lists and describes the modules. Figure 3 shows the prerequisite relationship between the modules.

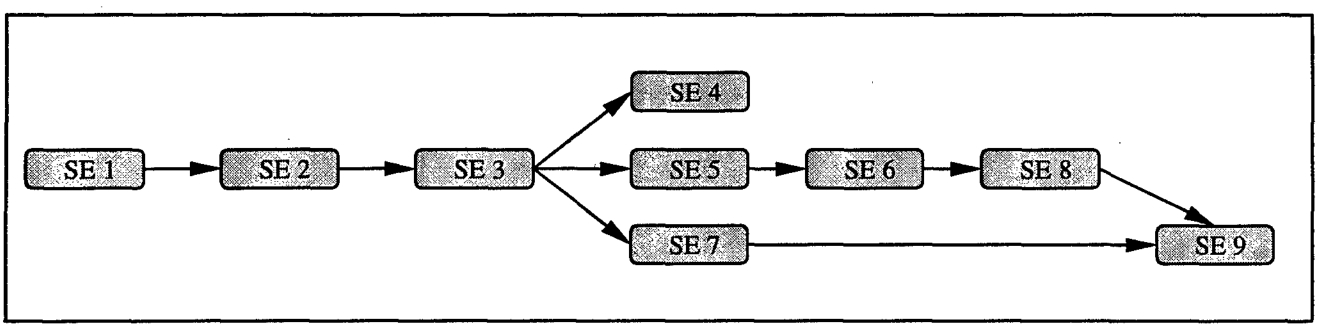


Figure 3 SE Module Prerequisite Chart

Each of the modules could be developed into a complete course. However, depending on the objectives and nature of a curriculum, some modules could be incorporated in a course with other material. For example, SE 5 and SE 6 might be combined to produce one course in "Software Analysis and Design."

Module	Title	Description
SE 1	Introduction to Computer Science for Software Engineers 1	This module provides an introduction to programming, usually with the procedural paradigm. The basic features of software engineering are integrated throughout the module.
SE 2	Introduction to Computer Science for Software Engineers 2	This module investigates data structures and the object-oriented paradigm while continuing with an introduction to software engineering.
SE 3	Introduction to Software Engineering	The module provides an overview of software engineering as a discipline; the module introduces students to the fundamental principles and methodologies of software engineering.
SE 4	Professionalism and Ethics	The module covers material on the historical, social, and economic issues in software engineering. It includes study of professional responsibilities, risks and liabilities, and intellectual property relative to the software engineering profession.
SE 5	Software Requirements	This module introduces basic concepts and principles of software requirements engineering, its tools and techniques, and methods for modeling software systems.
SE 6	Software Design	This module covers the methods and techniques used in the design of software systems. It includes architectural and detailed design, with an emphasis on object-oriented methods, the design process, and the design documentation and review.
SE 7	Software Quality	This module looks at how software quality assurance and configuration management are performed and how software process improvement is maintained in order to assure the highest possible quality.

Module	Title	Description
SE 8	Software Construction and Evolution	This module examines issues, methods, and techniques associated with constructing software, given a high-level design, and for maintaining software over its lifetime.
SE 9	Software Design Project	This module allows the students to use the skills gaining in previous modules as part of a development team on a semester-long software engineering project.

Table 2: SE Modules

The templates on the following pages provide more detail on the module content and information that would support course and curriculum design.

Module: SE 1	Introduction to Computer Science for Software Engineers 1
Prerequisite Knowledge	Pre-calculus and programming knowledge through loops in some language
Description	This module provides an introduction to programming usually with the procedural paradigm. The basic features of software engineering are integrated throughout the module.
Module Objectives	Upon completion of this module, students should be able to: • Implement programs using features such as arrays, strings, and records. • Perform elementary analysis and design tasks using the procedural paradigm. • Do individual inspections ("desk checking"). • Develop and implement a test plan.
Module Content	The module will include the following topics: • review of the most basic programming constructs (assignments, input/output, selection constructs, and loops) • more advanced programming constructs (one-dimensional arrays, strings, multi-dimensional arrays, text files, and records) • introduction to structured analysis and top-down design, including appropriate documentation • inspection and testing in the procedural paradigm
Recommended Module Format and Learning Activities	• four semester credits • three lecture hours per week • two to three laboratory hours per week (investigating programming features) • individual projects involving elementary software engineering techniques using the procedural paradigm

Module: SE 1	Introduction to Computer Science for Software Engineers 1
Resources and References	• Dale, Nell B.; Weems, Chip; & Headington, Mark. *Programming in C++*. Sudbury, Massachusetts: Jones and Bartlett, 1998. • Dale, Nell B. *Introduction to Turbo Pascal and Software Design*. Sudbury, Massachusetts: Jones and Bartlett, 1997. • Deitel, Harvey M. & Deitel, Paul J. *C++: How to Program*. Upper Saddle River, New Jersey: Prentice-Hall, 1997. • Deitel, Harvey M. & Deitel, Paul J. *Java: How to Program*. Upper Saddle River, New Jersey: Prentice-Hall, 1997. • Dale, Nell B. *A Laboratory Course in C++*. Sudbury, Massachusetts: Jones and Bartlett, 1997 (lab manual). • Humphrey, W. S. *Introduction to the Personal Software Process*. Reading, Massachusetts: Addison-Wesley, 1997.
Comments	The courses titled Introduction to Computer Science for Software Engineers 1 and 2 (CS1 and CS2) have been structured in the traditional manner for such courses in that the procedural paradigm is used in the first module, and the object-oriented paradigm in the second. A variation of this is to teach objects from the beginning, with procedural constructs being introduced as necessary. These modules differ from the traditional CS1 and CS2 courses in that the basics of software engineering (e.g., life cycle, inspections, configuration management, and quality assurance) are integrated into the various software development tasks undertaken.

Module: SE 2	Introduction to Computer Science for Software Engineers 2
Prerequisite Knowledge	SE 1
Description	This module investigates data structures and the object-oriented paradigm while continuing with an introduction to software engineering.
Module Objectives	Upon completion of this module, students should be able to: • Implement programs using features such as classes and pointers. • Write recursive programs. • Program using elementary data structures. • Perform analysis and design tasks using the object-oriented paradigm. • Work in a small team (four to six people) on the analysis, design, implementation, and testing of a project. • Do inspections and reviews within a team. • Do elementary configuration management and quality assurance within a small team.
Module Content	The module will include the following topics: • basic data structures (stacks, queues, binary trees) • introduction to recursion • introduction to object-oriented analysis and design, including appropriate documentation • abstract data types and classes • pointers and linked lists • inspection and testing in the object-oriented paradigm • introduction to configuration management and quality assurance
Recommended Module Format and Learning Activities	• four semester credits • three lecture hours per week • two to three laboratory hours per week (investigating data structures) • individual and team projects involving elementary software engineering techniques using the object-oriented paradigm

Module: SE 2	Introduction to Computer Science for Software Engineers 2
Resources and References	• Budd, Timothy. *Data Structures in C++ Using the Standard Template Library*. Reading, Massachusetts: Addison-Wesley, 1997. • Main, Michael & Savitch, Walter. *Data Structures and Other Objects Using C++*. Reading, Massachusetts: Addison-Wesley, 1997. • Main, Michael. *Data Structures and Other Objects Using Java*. Reading, Massachusetts: Addison-Wesley, 1998. • Robergé, James. *Data Structures in C++: A Laboratory Course*. Sudbury, Massachusetts: Jones and Bartlett, 1997 (lab manual).
Comments	The courses titled Introduction to Computer Science for Software Engineers 1 and 2 (CS1 and CS2) have been structured in the traditional manner for such courses in that the procedural paradigm is used in the first module, and the object-oriented paradigm in the second. A variation of this is to teach objects from the beginning, with procedural constructs being introduced as necessary. These modules differ from the traditional CS1 and CS2 courses in that the basics of software engineering (e.g., life cycle, inspections, configuration management, and quality assurance) are integrated into the various software development tasks undertaken.

Module: SE 3	Introduction to Software Engineering
Prerequisite Knowledge	SE 2
Description	This module provides an overview of software engineering as a discipline; the module introduces students to the fundamental principles and methodologies of software engineering. It covers basic knowledge about software requirements, software design, software construction, software management, and software quality. It provides minimum prerequisite knowledge for more detailed and specialized study of software engineering. Students gain experience, via a team project, about life-cycle development of software systems.
Module Objectives	Upon completion of this course, students should be able to: • Identify and discuss the technical and engineering activities of producing a software product. • Describe issues, principles, methods and technology associated with software engineering theory and practices (e.g., planning, requirements engineering, design, coding, testing, quality assurance, and configuration management). • Working as part of a team, use a software development process to develop a software product.
Module Content	The module will include the following topics: • introduction to software engineering • models of the software development process • project planning and organization • software requirements and specifications • software design techniques • software quality assurance • software testing • software tools and environments • team project activities
Recommended Module Format and Learning Activities	• three to four semester credits • three lecture/discussion hours per week • two to three laboratory hours per week - team process.and team project work (status reports, team meetings, reviews and inspections, tests and demonstrations)

Module: SE 3	Introduction to Software Engineering
Resources and References	• Pressman, Roger S. *Software Engineering: A Practitioner's Approach*, 4th ed. New York, New York: McGraw-Hill, 1997. • Sommerville, I. *Software Engineering*, 5th ed. Reading, Massachusetts: Addison-Wesley, 1995. • Pfleeger S. *Software Engineering Theory and Practice*. Upper Saddle River, New Jersey: Prentice-Hall, 1998. • Humphrey, W. S. *Introduction to the Team Software Process*. Reading, Massachusetts: Addison-Wesley, 1999. • Brooks, F.P. *The Mythical Man-Month, Essays on Software Engineering*, anniversary ed. Reading, Massachusetts: Addison-Wesley, 1995. • Marciniak, John J. *Encyclopedia of Software Engineering*. New York, New York: John Wiley & Sons, Inc., 1994. • Dorfman, M. & Thayer, R., eds. *Software Engineering*. Los Alamitos, California: IEEE Computer Society Press, 1997.

Module: SE 4	Professionalism and Ethics
Prerequisite Knowledge	SE 3
Description	This module covers material on the historical, social, and economic issues in software engineering. It includes the study of professional responsibilities, risks and liabilities, and intellectual property relative to the software engineering profession.
Module Objectives	Upon completion of this module, students should be able to: • Describe and discuss historical, social, economic, ethical, and professional issues related to the discipline of software engineering • Identify key sources for information and opinion about professionalism and ethics. • Analyze, evaluate, and assess ethical and professional computing and software engineering case studies.
Module Content	The module will include the following topics: • historical, social, and economic context of software engineering • definitions of software engineering subject areas and professional activities • professional societies • professional ethics • professional competency and life-long learning • uses, misuses, and risks of software • information security and privacy • business practices and the economics of software • intellectual property and software law • social responsibilities
Recommended Module Format and Learning Activities	• three semester credits • three lecture/discussion hours per week • presentation of ethical and professional case studies in computing and software engineering • individual projects: – write an expository paper on an historical, social, economic, or professional issue in software engineering – write a position on a module case study (on an ethical or professional issue)

Module: SE 4	Professionalism and Ethics
Resources and References	• Edgar, S.L. *Morality and Machines: Perspectives on Computer Ethics*. Sudbury, Massachusetts: Jones and Bartlett, 1996. • Epstein, R. G. *The Case of the Killer Robot*. New York, New York: John Wiley, 1997. • ACM/IEEE-CS Task Force, *Software Engineering Code of Ethics and Professional Practice Project* [online]. Available WWW <URL: htp://www.acm.org/serving/se/SWCEPP.htm> (1999). • Boehm, B. W. *Software Engineering Economics*. Upper Saddle River, New Jersey: Prentice-Hall, 1981.

Module: SE 5	Software Requirements
Prerequisite Knowledge	SE 3 Introduction to Software Engineering
Description	This module introduces basic concepts and principles of software requirements engineering, its tools and techniques, and methods for modeling software systems. Various approaches to requirements analysis are examined; structured, object-oriented, and formal approaches are studied.
Module Objectives	Upon completion of this module, students should be able to: • Describe the role of requirements engineering within the software life cycle. • Describe, compare and contrast, and evaluate structured, object-oriented, data-oriented, and formal approaches to requirements modeling. • Gather the requirements necessary to develop the specifications, given a "customer" who wants a software system to be developed. • Perform an analysis, do some planning and risk assessment, and develop an informal requirements specification, given a set of requirements. • Model, prototype, and specify requirements for a software system. • Review and inspect software requirements.

Module: SE 5	Software Requirements
Module Content	The module will include the following topics: • software life-cycle models • introduction to project planning and risk management • requirements modeling and analysis • Software requirements specification • Software requirements elicitation and analysis • Structured methods • object-oriented methods • formal methods in requirements (formal and executable specifications) • requirements verification and validation • requirements elicitation (e.g., scripting, development of use cases and interface) • software requirements metrics • prototyping user interfaces • customer acceptance of requirements
Recommended Module Format and Learning Activities	• one to three semester credits • three lecture/discussion hours per week • a team or individual project to collect, prototype, model, specify, and verify the requirements for a software system
Resources and References	• Davis, A. *Software Requirements: Objects, Functions, & States*. Upper Saddle River, New Jersey: Prentice Hall, 1993. • Jackson, M. *Software Requirements & Specifications: A Lexicon of Practice, Principles and Prejudices*. Reading, Massachusetts: Addison-Wesley, 1995. • Dorfman, M., Thayer, R., & Davis, A., eds. *Software Requirements Engineering,* 2nd ed. Los Alamitos, California: IEEE Computer Society Press, 1997. • Dorfman, M. & Thayer, R., eds. *Software Engineering*. Los Alamitos, California: IEEE Computer Society Press, 1997. • Loucopoulos, P. & Karakostas, V. *Systems Requirements Engineering*. New York, New York: McGraw-Hill, 1995. • Marciniak, John J. *Encyclopedia of Software Engineering*. New York, New York: John Wiley & Sons, Inc., 1994.

Module: SE 6	Software Design
Prerequisite Knowledge	SE 3, SE 5
Description	This module covers the methods and techniques used in the design of software systems. It includes architectural and detailed design, with an emphasis on object-oriented methods, the design process, and the design documentation and review.
Module Objectives	Upon completion of this module, students should be able to: • Prototype a user interface, given a set of interface requirements. • Perform an object-oriented architectural design in a team of three to five people, given a requirements specification document developed by someone else. • Perform a partial design and prototyping in one or more iterations, given a requirements specification document developed by someone else. • Develop a detailed design, given a preliminary design document. • Code a design in a particular language, given a detailed design.
Module Content	The module will include the following topics: • introduction to software architecture • design patterns • object-oriented architectural design • experimentation in design • design prototyping • working on a design team • detailed design and implementation issues • design reviews (preliminary and detailed) • using the design document for coding
Recommended Module Format and Learning Activities	• one to three semester credits • three lecture hours per week for each module • individual and team projects

Module: SE 6	Software Design
Resources and References	• Booch, Grady. *Object-Oriented Analysis and Design with Applications,* 2nd ed. Redwood City, California: Benjamin/Cummings, 1994. • Easteal, Charles & Davies, Gordon. *Software Engineering Analysis and Design.* New York, New York: McGraw-Hill, 1989. • Shaw, Mary & Garlan, David. *Software Architecture: Perspectives on an Emerging Discipline.* Upper Saddle River, New Jersey: Prentice-Hall, 1996. • Bass, Len; Clements, Paul; & Kazman, Rick. *Software Architecture in Practice.* Reading, Massachusetts: Addison-Wesley, 1998. • Jacobson, Ivar et al. *Object-Oriented Software Engineering: A Use-Case Driven Approach.* Reading, Massachusetts: Addison-Wesley, 1992.

Module: SE 7	Software Quality
Prerequisite Knowledge	SE 3
Description	This module looks at how software quality assurance and configuration management are performed and how software process improvement is maintained in order to assure the highest possible quality.
Module Objectives	Upon completion of this module, students should be able to: • Understand the role and importance of software quality assurance in a software project. • Understand the role and importance of configuration management in a software project. • Understand how software metrics are developed and used to insure quality. • Develop a quality assurance plan. • Develop a configuration management plan. • Perform reviews, inspections, and audits. • Understand how a configuration control board works. • Understand how the major methods for software process improvement work.
Module Content	The module will include the following topics: • software quality assurance • software quality metrics • software configuration management • software verification and validation (V&V) • reviews, inspections, and audits • configuration control boards • software process improvement models
Recommended Module Format and Learning Activities	• three semester credits • three lecture hours per week • individual and team projects

Module: SE 7	Software Quality
Resources and References	• Freedman, Daniel & Weinberg, Gerald. *Handbook of Walkthroughs, Inspections, and Technical Reviews*, 3rd ed. New York, New York: Dorset House, 1990. • Ayer, Steve & Patrinostro, Frank. *Software Configuration Management*. New York, New York: McGraw-Hill, 1992. • Humphrey, Watts S. *A Discipline for Software Engineering*. Reading, Massachusetts: Addison-Wesley, 1995. • Paulk, M.; Weber, C.; Curtis, B.; & Chrissis, M. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Reading, Massachusetts: Addison-Wesley, 1995. • Schmauch, Charles H. *ISO 9000 for Software Developers,* revised ed. Milwaukee, Wisconsin: ASQ Quality Press, 1995.

Module: SE 8	Software Construction & Evolution
Prerequisite Knowledge	SE 2, SE 3, SE 6 course/module in Data Structures and Algorithms
Description	This module examines issues, methods, and techniques associated with constructing software, given a high-level design, and for maintaining software over its lifetime.
Module Objectives	Upon completion of this module, students should be able to: • Use programming languages, software design knowledge, and construction tools to implement a high-level design. • Use and analyze an individual software process in constructing a software module or unit. • Use software implementation tools to construct software. • Analyze the impact of the design and construction process on long-term software maintainability and evolution. • Describe and discuss maintenance processes and techniques.

Module: SE 8	Software Construction & Evolution
Module Content	The module includes the following topics: • software construction overview • detailed design • individual software process and metrics • programming language issues • implementation tools • coding standards and styles • review and testing • software reuse • maintenance overview • maintenance process and metrics • maintenance techniques
Recommended Module Format and Learning Activities	• one to three semester credits • three lecture/discussion hours per week • individual projects involving unit/module development and maintenance (design, coding, review, testing)
Resources and References	• Humphrey, W. S. *A Discipline for Software Engineering*. Reading, Massachusetts: Addison-Wesley, 1995. • Meyer, B. *Object-Oriented Software Construction*. Upper Saddle River, New Jersey: Prentice Hall, 1997. • Deimel L. & Naveda, F. *Reading Computer Programs: Intructor's Guide and Exercises* (CMU/SEI-90-EM-3). Pittsburgh, Pa: Software Engineering Institute, Carnegie Mellon University, August 1990. • Pigoski, T.M. *Practical Software Maintenance*. New York, New York: John Wiley & Sons, 1997. • Dijkstra, E. *A Discipline of Programming*. Upper Saddle River, New Jersey: Prentice-Hall, 1976. • Marciniak, John J. *Encyclopedia of Software Engineering*. New York, New York: John Wiley & Sons, Inc., 1994. • Dorfman, M. & Thayer, R., eds. *Software Engineering*. Los Alamitos, California: IEEE Computer Society Press, 1997.

Module: SE 9	Senior Design Project
Prerequisite Knowledge	SE 3, SE 5, SE 6, SE 7, SE 8 Domain-specific knowledge may be needed depending on the project.
Description	This module allows the students to use the skills gained in previous modules as part of a development team on a semester-long software engineering project.
Module Objectives	Upon completion of this module, students should be able to: • Understand the major aspects of software project management. • Work on a large team project. • Work for a client on a project that will actually be used. • Make oral and written project presentations.
Module Content	The module will include the following topics: • project planning and risk analysis • project tracking and scheduling • large-team project roles and structure • team and client presentations
Recommended Module Format and Learning Activities	• four semester credits • The number of lecture hours per week can vary and would include team presentations at various points during the semester. • a single semester-long team project
Resources and References	• Brooks, Frederick. *The Mythical Man-Month,* 20th anniversary ed. Reading, Massachusetts: Addison-Wesley, 1995. • Humphrey, W. S. *Introduction to the Team Software Process*. Reading, Massachusetts: Addison-Wesley, 1999. • Yourdon, Edward. *Death March: The Complete Software Developer's Guide to "Mission Impossible" Projects.* Upper Saddle River, New Jersey: Prentice-Hall, 1997.

5.6 Sample Curriculum Implementations

This section contains a description of a sample curriculum in software engineering and descriptions of three software engineering focused curricula in computer science, information systems, and computer engineering.

A. Sample Curriculum in *Software Engineering*

i) Curriculum Objectives

This curriculum leads to a BS degree in Software Engineering. A student completing this curriculum would be prepared for the following:

- To be employed as an entry-level software engineer
- To pursue graduate work in computer science (with additional undergraduate courses in computer science)
- To pursue graduate work in software engineering

ii) Curriculum Design

(1) The sample curriculum was designed so that it could be adapted to programs that wish to evolve from a "traditional" computer science program into a program with more emphasis on software engineering.

(2) The sample curriculum is based upon the design concepts set down earlier in this document. Its design is influenced by the curriculum guidance provided in *Computing Curricula 1991* [Tucker 91] and *Engineering Criteria 2000: Criteria for Accrediting Engineering Programs* [ABET].

(3) The curriculum includes a total of 120 semester credit hours distributed as follows:

Area	**Credit Hours**
Computer Science - required	21
Software Engineering - required	24
Computer Science / Software Engineering - electives	9
Lab Science	12
Engineering Sciences (other than CS)	9
Mathematics (including discrete mathematics)	24
Communication/Humanities/Social Sciences	18
open electives	3
total	120

(4) This 120 semester-hour curriculum was designed both to satisfy the growing movements towards government constraints on credit hours while still satisfying ABET and licensing exam requirements. If more than 120 hours is allowed in the curriculum, additional requirements might include the substitutions of several courses in the place of Software Systems 1 and 2 and Engineering Sciences 1 and 2 that are listed below in the curriculum schedule.

iii) The next section contains a semester-by-semester schedule of the courses that make up the sample curriculum. The following comments elaborate on the courses and schedule:

(1) The one-credit Software Engineering Seminar course would provide a "breath-first" overview of the software engineering discipline. It would also include an introduction to professional ethics.

(2) The courses titled Introduction to Computer Science for Software Engineers 1 and 2 would be delivered with two hours of lecture per week and a two to three hour closed lab. Software engineering concepts and practices would be introduced in these two courses. The lab would provide the opportunity to introduce students, at an early stage, to disciplined engineering practices.

(3) The Introduction to Software Engineering course would be delivered with two hours of lecture per week and a two to three hour closed lab. The lab would introduce students to a team software process to be used in a team project.

(4) The courses titled Engineering Sciences 1 and 2 would provide an overview of the following topics in the engineering sciences: dynamics, electric circuits, fluid mechanics, material science, mechanics of materials, statics, and thermodynamics.

(5) The courses titled Software Systems 1 and 2 would provide an overview of various software systems: database systems, operating systems, compilers and translators, and concurrent and distributed systems.

(6) The courses titled Software Analysis and Design 1 and 2 would educate students in the concepts and practices used in the requirements analysis and design of a large software system.

(7) The Senior Design Project course would involve the development of a software product for a real customer. The course deliverables (project plans, requirements and design specifications, test plans, source code, inspection reports, and user/reference documents) would be used to assess the software engineering objectives of the curriculum.

iv) Curriculum Schedule

	Courses	Credit		Courses	Credit
Sem 1	SE - SE Seminar	1	Sem 2	CS - Intro to CS for SE 2	3
	CS - Intro to CS for SE 1	3		MA - Discrete Math	3
	MA - Calculus 1	4		MA - Calculus 2	4
	SC - Chemistry 1	4		HU/SS elective	3
	HU - Communications 1	3		HU - Communications 2	3
	total	15		total	16
Sem 3	CS - Data Structure & Algorithms	3	Sem 4	SE - Intro to Software Engineering	4
	CS - Computer Organization	3		MA - Differential Equations	3
	MA - Calculus 3	4		CS - Programming Languages	3
	SC - Physics 1	4		SC - Physics 2	4
	total	14		total	14
Sem 5	SE - Software Anal. & Design 1	3	Sem 6	SE - Formal Methods	3
	MA - Probability and Statistics	3		SE - Software Quality	3
	CS - Software Systems 1	3		CS - Software Systems 2	3
	HU - Technical Writing	3		EN - Engineering Economics	3
	EN - Engineering Sciences 1	3		EN - Engineering Sciences 2	3
	total	15		total	15
Sem 7	SE - Software Anal. & Design 2	3	Sem 8	SE - Senior Design Project	4
	CS/SE elective	3		CS/SE elective	6
	MA - Linear Algebra	3		open elective	3
	SE - Professional Ethics	3		HU/SS elective	3
	HU/SS elective	3			
	total	15		total	16
total credits = 120					

B. A Sample Curriculum in Computer Science with an emphasis in Software Engineering

i) Curriculum Objectives

This curriculum leads to a BS degree in Computer Science with a prescribed part devoted to the study and practice of software engineering. A student completing this curriculum would be prepared for the following:

- To be employed as an entry-level software engineer
- To pursue graduate work in computer science
- To pursue graduate work in software engineering

ii) Curriculum Design

(1) The sample curriculum was designed so that it could be adapted to programs that wish to evolve from a "traditional" computer science program into a program with more emphasis on software engineering. To achieve the above objective of "be employed as an entry-level software engineer," additional courses in software engineering may be required.

(2) The sample curriculum is based upon the design concepts set down earlier in this document. Its design is influenced by the curriculum guidance provided in *Computing Curricula 1991* [Tucker 91] and *CSAC/CSAB Criteria 2000* [CSAB].

(3) The curriculum includes a total of 120 semester credit hours distributed as follows:

Area	**Credit Hours**
Computer Science - required	24
Software Engineering - required	9
Computer Science / Software Engineering - electives	9
Science	12
Mathematics	18
Communication/Humanities/Social Sciences	30
open electives	12
total	120

iii) The next section contains a semester-by-semester schedule of the courses that make up the sample curriculum. The following comments elaborate on the courses and schedule:

(1) The one-credit Introduction to Computing course would provide a "breath-first" overview of the computing discipline. It would also include an introduction to professional ethics.

(2) The courses titled Computer Science 1 and 2 would be delivered with two hours of lecture per week and a two to three hour closed lab. Software engineering concepts and practices would be introduced in these two courses. The lab would provide the opportunity to introduce students, at an early stage, to disciplined engineering practices.

(3) The Introduction to Software Engineering course would be delivered with two hours of lecture per week and a two to three hour closed lab. The lab would introduce students to a team software process to be used in a team project.

(4) The Software Analysis and Design course would educate students in the concepts and practices used in the requirements analysis and design of a large software system.

(5) The Senior Design Project course would involve the development of a software product for a real customer. The course deliverables (project plans, requirements and design specifications, test plans, source code, inspection reports, and user/reference documents) would be used to assess the software engineering objectives of the curriculum.

v) Curriculum Schedule

BS in Computer Science (emphasis in Software Engineering)

	Courses	Credit		Courses	Credit
Sem 1	CS - Intro to Computing	1	Sem 2	CS - Computer Science 2	3
	CS - Computer Science 1	3		MA - Discrete Math	3
	MA - Calculus 1	4		MA - Calculus 2	4
	HU - Communications 1	3		SC - Physics 1	4
	HU/SS elective	3		HU/SS elective	3
	total	14		total	17
Sem 3	CS - Data Structure & Algorithms	3	Sem 4	SE - Intro to Software Engineering	3
	CS - Computer Organization	3		MA - math elective	4
	SC - Physics 2	4		SC - science elective	4
	HU - Communications 2	3		HU/SS elective	3
	HU/SS elective	3			
	total	16		total	14
Sem 5	CS - Programming Languages	3	Sem 6	CS - Formalisms & Computation	3
	CS/SE elective	3		SE - Software Analysis & Design	3
	MA - Probability & Statistics	3		open elective	6
	open elective	3		HU/SS elective	3
	HU – Technical Writing	3			
	total	15		total	15
Sem 7	CS - Concurrent/ Distributed Sys	3	Sem 8	SE - Senior Design project	3
	CS/SE elective	3		CS/SE elective	3
	MA - math elective	3		MA - math elective	3
	CS - Professional Ethics	2		open elective	3
	HU/SS elective	3		HU/SS elective	3
	total	14		total	15
total credits = 120					

C. A Sample Curriculum in Information Systems with an emphasis in Software Engineering

i) Curriculum Objectives

This curriculum leads to a BS degree in Information Systems with a prescribed part devoted to the study and practice of software engineering. A student completing this curriculum would be prepared for the following:

- To be employed as an entry-level software engineer for information systems
- To pursue graduate work in information systems
- To pursue graduate work in software engineering (perhaps with some additional computer science or mathematics courses as prerequisite to program entry)

ii) Curriculum Design

(1) The sample curriculum was designed so that it could be adapted to programs that wish to evolve from a "traditional" information systems program into a program with more emphasis on software engineering. To achieve the above objective of "be employed as an entry-level software engineer," additional courses in software engineering may be required.

(2) The sample curriculum is based upon the design concepts set down earlier in this document. Its design is influenced by the curriculum guidance provided in *IS'97: Model Curriculum and Guidelines and Undergraduate Degree Programs in Information Systems* [Davis 97] and the draft accreditation criteria for information systems degrees [Gorgone 99].

(3) The curriculum includes a total of 120 semester credit hours distributed as follows:

Area	**Credit Hours**
Information Systems and Software Engineering	32
Humanities	24
Business	18
Computer Science	12
Mathematics and Statistics	14
Social Science	12
Natural Science	8
total	120

iii) The next section contains a semester-by-semester schedule of the courses that make up the sample curriculum. The following comments elaborate on the courses and schedule:

(1) The courses titled Computer Science 1 and 2 would be delivered with two hours of lecture per week and a two to three hour closed lab. Software engineering concepts and practices would be introduced in these two courses. The lab would provide the opportunity to introduce students, at an early stage, to disciplined engineering practices.

(2) The courses titled Information Systems 1 and 2 would be delivered with three hours of lecture per week and a one to two hour closed lab. The lab would introduce students to elements of information technology including personal computing applications, programming, databases, and Internet technologies.

(3) The Requirements Analysis course would educate students in the concepts and practices used in the specification and design of a large software system. The course would cover the diagrammatic formalisms of structured or object-oriented methods.

(4) The Design Project course would involve the development of a software product for a real customer. The course deliverables (project plans, requirements and design specifications, test plans, source code, inspection reports, and user/reference documents) would be used to assess the software engineering objectives of the curriculum.

v) Curriculum Schedule

BS in Information Systems (emphasis in Software Engineering)

	Courses	Credit		Courses	Credit
Sem 1	IS - Information Systems 1	4	Sem 2	IS - Information Systems 2	4
	BS - Economics 1	3		BS - Economics 2	3
	MA - Calculus 1	4		MA - Calculus 2	4
	HU - Communications 1	3		HU - Communications 2	3
	total	14		total	14
Sem 3	IS - Database Management	3	Sem 4	IS - Requirements Analysis	3
	CS - Computer Science 1	3		CS - Computer Science 2	3
	MA - Discrete Math	3		BS - Financial Accounting	3
	SC - Natural Science 1	4		SC - Natural Science 2	4
	SS - General Psychology	3		SS - Cognitive Psychology	3
	total	16		total	16
Sem 5	IS - Human-Comp. Interaction	3	Sem 6	IS - Software Engineering	3
	CS - elective	3		CS - elective	3
	BS - Applied Statistics	3		SS - Research Methods	3
	BS - elective	3		HU - Logic	3
	HU - Techniques of Speaking	3		HU - Technical Writing	3
	total	15		total	15
Sem 7	IS - Design Project	3	Sem 8	IS - Design Project	3
	IS - Distrib. Computing & Networking	3		IS - elective	3
	HU - Critical Reasoning	3		HU - Computer Ethics	3
	BS - Organizational Behavior	3		SS - elective	3
	BS - elective	3		HU - elective	3
	total	15		total	15
Total credits = 120					

D. A Sample Curriculum in Computer Engineering with an emphasis in Software Engineering

i) Curriculum Objectives

This curriculum leads to a BS degree in Computer Engineering with a prescribed part devoted to the study and practice of software engineering. A student completing this curriculum would be prepared for the following:

- To be employed as an entry-level software engineer
- To pursue graduate work in computer engineering
- To pursue graduate work in software engineering

ii) Curriculum Design

(1) The sample curriculum was designed so that it could be adapted to programs that wish to evolve from a "traditional" computer engineering program into a program with more emphasis on software engineering. To achieve the above objective of "be employed as an entry-level software engineer," additional courses in software engineering may be required.

(2) The sample curriculum is based upon the design concepts set down earlier in this document. Its design is influenced by the curriculum guidance provided in *Computing Curricula 1991: Report of the ACM/IEEE-CS Joint Curriculum Task Force* [Tucker 91], the *CSAC/CSAB Criteria 2000: Criteria for Accrediting Programs in Computer Science in the United States* [CSAB], and the *Engineering Criteria 2000: Criteria for Accrediting Engineering Programs* [ABET].

(3) The curriculum emphasizes digital hardware and embedded software systems.

(4) The sample curriculum was designed to insure that it meets the minimal ABET curriculum content requirements:

(a) one year of math/basic science (32 hours)

(b) one-half year of humanities/social sciences (16 hours)

(c) one and one-half year of engineering topics (1 course outside of the major disciplinary area) (48 hours)

(5) A typical computer engineering program does not seem to exist. Some programs have more computing courses than electrical engineering and vice versa. Since a wide variation exists, the curriculum given below is not meant as a rigid template (i.e., it is open to change depending upon the goals of the curriculum designers).

The curriculum includes a total of 124 semester credit hours distributed as follows:

Area	Credit Hours
Electrical Engineering - required	30
Electrical Engineering - electives	3
Computer Science - required	15
Software Engineering - required	9
Software Engineering - electives	3
Science	12
Mathematics	21
Communication/Humanities/Social Sciences	24
Engineering/Engineering Science	5
total	122

iii) The next section contains a semester-by-semester schedule of the courses that make up the sample curriculum. The following comments elaborate on the courses and schedule:

(1) The two-credit Introduction to Engineering course would provide a "breath-first" overview of the engineering discipline. It would also include an introduction professional ethics.

(2) The courses titled Computer Science 1 and 2 would be delivered with two hours of lecture per week and a two to three hour closed lab. Software engineering concepts and practices would be introduced in these two courses. The lab would provide the opportunity to introduce students, at an early stage, to disciplined engineering practices.

(3) The Introduction to Software Engineering course would be delivered with two hours of lecture per week and a two to three hour closed lab. The lab would introduce students to a team software process to be used in a team project.

(4) The Software Analysis and Design course would educate students in the concepts and practices used in the requirements analysis and design of a large software system.

(5) The Embedded Systems course would emphasize the development of real-time software for computers embedded in systems. The concepts of synchronization, schedulability analysis, and fundamental control theory, would be studied.

(6) The EE Project Lab courses would ideally need to focus on capstone computer engineering projects at the senior level. Where software is involved, appropriate software engineering principles would be employed.

iv) Curriculum Schedule

BS in Computer Engineering (emphasis in Software Engineering)

	Courses	Credit		Courses	Credit
Sem 1	ER - Intro to Engineering	2	Sem 2	EE - Fundamentals	3
	CS - Computer Science 1	3		CS - Computer Science 2	3
	MA - Calculus 1	3		MA - Calculus 2	3
	HU - Communications 1	3		HU - Communications 2	3
	SC - Chemistry 1	4		HU/SS elective	3
	total	15		total	15
Sem 3	EE - Digital Systems	3	Sem 4	EE - Digital Design	3
	CS - Discrete Math	3		SE - Intro to Software Engineering	3
	MA - Calculus 3	3		MA - Linear Algebra	3
	SC - Physics 1	4		SC - Physics 2	4
	HU/SS elective	3		MA - Differential Equations	3
	total	16		total	16
Sem 5	EE - Project Lab 1	3	Sem 6	EE - Project Lab 2	3
	SE - Software Analy. & Design	3		SE - Embedded Systems	3
	MA - Probability and Statistics	3		CS - Algorithms	3
	EE - Electronics 1	3		EE - Electronics 2	3
	ER - Thermodynamics	3		EE - Communication Systems	3
	total	15		total	15
Sem 7	EE - Project Lab 3	3	Sem 8	EE - Project Lab 4	3
	MA - Numerical Analysis	3		SE elective	3
	EE/CS - Computer Architecture	3		EE elective	3
	HU/SS elective	3		HU/SS elective	3
	HU/SS elective	3		HU/SS elective	3
	total	15		total	15
total credits = 122					

6 Curriculum Support

6.1 Faculty

A high-quality faculty and staff is the single most critical element in the success of a program. Faculty needs both advanced education in computing and experience in software engineering practice. Because of the dynamic nature of computing, it is essential that faculty continue to engage in professional development (research, participation in professional societies, consulting, and technical training).

6.2 Infrastructure

The program must provide adequate infrastructure and technical support. This includes well-equipped laboratories and classrooms, modern CASE Tools, and sufficient reference and documentation material.

6.3 Industry Participation

A critical element in the success of a software engineering curriculum is the involvement and participation of industry. Industrial advisory boards and industry-academic partnerships help maintain curriculum relevance and currency.

6.4 Student Involvement

Interaction with students about curriculum development and delivery provides valuable information for assessing and analyzing a curriculum. Involvement of students in professional organizations and activities extends and enhances their education.

7 Curriculum Assessment and Accreditation

In order to maintain a quality curriculum, a software engineering focused program should be assessed on a regular schedule. The assessment should evaluate the objectives, curriculum content, curriculum delivery, and outcomes of the program. This is best accomplished in conjunction with a recognized accreditation organization. In the United States ABET and CSAB provide curriculum guidance and accreditation criteria for the accreditation of engineering and computing programs [ABET, CSAB]. A joint IEEE-CS/ACM committee has developed draft accreditation criteria for software engineering programs [Barnes 98].

8 Conclusion

Software engineering is a maturing discipline that is becoming increasingly critical in all aspects of human endeavor. The demand for well-educated software engineers is increasing, but sufficient computing programs to support this demand do not exist. We believe that these Guidelines will help solve this problem by providing information and guidance that will assist faculty and institutions in creating and implementing quality software engineering courses and curricula.

9 Acknowledgments

Our special thanks go to Dr. Nancy Mead of the SEI for her leadership of the WGSEET, her substantial work in promoting efforts to improve software engineering education, and her encouragement and support for these Guidelines.

We would also like to acknowledge the help of the following individuals in formulating and/or reviewing this document: Neal Coulter (University of North Florida), Anthony Cowling (University of Sheffield), Jorge Diaz-Herrera (Southern Polytechnic University), Robert Dupuis (University of Quebec at Montreal), Peter Knoke (University of Alaska), Jim McDonald (Monmouth University), Melody Moore (Georgia Institute of Technology), Fernando Naveda (Rochester Institute of Technology), Dale Oexmann (Rose-Hulman Institute of Technology), Michael Ryan (Dublin City University), and Laurie Werth (University of Texas).

For further information about the development of this document or comments on its content, please contact

Thomas B. Hilburn
Department of Computing & Mathematics
Embry-Riddle Aeronautical University
Daytona Beach, Fl 32114
1-904-226-6889
hilburn@db.erau.edu

10 References

[ABET] Accreditation Board for Engineering and Technology. *Engineering Criteria 2000: Criteria for Accrediting Engineering Programs* [online]. Available WWW <URL: http://www.abet.org/> (1998).

[Barnes 98] Barnes et al. "Draft Software Engineering Accreditation Criteria." *Computer 31*, 4 (May 1998): 73-75.

[BCS 89] British Computer Society and the Institution of Electrical Engineers. *A Report on Undergraduate Curricula for Software Engineering*. London, England: Institution of Electrical Engineers, June 1989.

[Cowling 98] Cowling, A.J. "A Multi-dimensional Model of the Software Engineering Curriculum, " 44-55. *Proceedings of the 11th Conference on Software Education & Training*. Atlanta, Georgia, February 22-25, 1998. Los Alamitos, California: IEEE Computer Society Press, 1998.

[CSAB] Computing Sciences Accreditation Board, Inc. *CSAC/CSAB Criteria 2000: Criteria for Accrediting Programs in Computer Science in the United States*, Version 0.8 [online]. Available WWW <URL: http://www.csab.org/> (1999).

[Davis 97] Davis et. al. *IS'97: Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems* [online]. Available WWW <URL: http://www.acm.org/education/curricula.html> (1997).

[Denning 92] Denning, P.J. "Educating a New Engineer" *Communications of the ACM 35*, 12 (December 1992): 83-97.

[Diaz-Herrera 99] Diaz-Herrera, J. L. *A Profile of Software Engineers*, (individual communication), May 1999.

[Dupuis 98] Dupuis et al. *A Guide to the Software Engineering Body of Knowledge, A Straw Man Version*. Los Alamitos, California: IEEE Computer Society, September 1998.

[Ford 96] Ford, G. & Gibbs, N.E. *Mature Profession of Software Engineering* (CMU/SEI-96-TR-004, ESC-TR-96-004). Pittsburgh, Pa: Software Engineering Institute, Carnegie Mellon University [online]. WWW <URL: http://www.sei.cmu.edu/publications/documents/96.reports/96.tr.004.html> (1996).

[Ford 94] Ford, G. *A Progress Report on Undergraduate Software Engineering Education* (CMU/SEI-94-TR-011, ESC-TR-94-011). Pittsburgh, Pa: Software Engineering Institute, Carnegie Mellon University, 1994 [online]. Available WWW <URL: http://www.sei.cmu.edu/publications/documents/94.reports/94.tr.011.html> (1994).

[Ford 90] Ford, G A. *1990 SEI Report on Undergraduate Software Engineering Education* (CMU/SEI-90-TR-003, ADA223881). Pittsburgh, Pa: Software Engineering Institute, Carnegie Mellon University, March 1990 [online]. Available WWW <URL: http://www.sei.cmu.edu/publications/documents/90.reports/90.tr.003.html> (1990).

[Garlan 97] Garlan, D.; Gluch, D. P.; & Tomayko, J. E. "Agents of Change: Educating Software Engineering Leaders." *IEEE Computer 30,* 11 (November 1997): 59-65.

[Gibbs 94] Gibbs, W.W. "Software's Chronic Crisis." *Scientific American 271*, 3 (September 1994): 86-95.

[Gorgone 99] Gorgone, John. *Draft Criteria for Accrediting Programs in Information Systems* [online]. Available WWW <URL: http://www.acm.org/education/curricula.html#IS97 > (1999).

[Hilburn 99] Hilburn et al. *A Software Engineering Body of Knowledge*, Version 1.0 (CMU/SEI-99-TR-004, ESC-TR-99-004). Pittsburgh, Pa: Software Engineering Institute, Carnegie Mellon University, April 1999 [online]. Available WWW <URL: http://www.sei.cmu.edu/publications/documents/99.reports/99tr004/99tr004abstract.html> (1999).

[Hilburn 97] Hilburn, T. B. "SE Education: A Modest Proposal." *IEEE Software 14*, 6 (November 1997): 44-48.

[IEEE-CS 98] IEEE-CS/ACM Joint Task Force on Software Engineering Ethics and Professional Practices. *Software Engineering Code of Ethics and Professional Practice* [online]. Available WWW <URL: http://www.computer.org/tab/seprof/code.htm> (1999).

[IEEE-CS 97] IEEE Computer Society, *Software Engineering Standards Collection*, 1997 ed. Los Alamitos, California: IEEE Computer Society Press, 1997.

[Lethbridge 98] Lethbridge, T.C. "Survey of the Relevance of Computer Science and Software Engineering Education," 44-55. *Proceedings of the 11th Conference on Software Education & Training*. Pittsburgh, Pennsylvania, February 22-25, 1998. Los Alamitos, California: IEEE Computer Society Press, 1998.

[Moore 94] Moore, Melody & Potts, Colin. "Learning by Doing: Goals and Experiences of Two Software Engineering Project Courses," 151-164. *Proceedings of the Seventh Software Engineering Institute Conference on Software Engineering Education.* San Antonio, Texas, January 6, 1994. Berlin, Germany: Springer-Verlag, 1994.

[Naveda 98] Naveda, J. & Lutz, M. "Crafting a Baccalaureate Program in Software Engineering," 74-80. *Proceedings of the 10th Conference on Software Education & Training.* Virginia Beach, Virginia, April 22-25, 1998. Los Alamitos, California: IEEE Computer Society Press, 1998.

[Tucker 91] Tucker, A. B., ed. *Computing Curricula 1991: Report of the ACM/IEEE-CS Joint Curriculum Task Force*. Los Alamitos, California: IEEE Computer Society Press, 1991.

[WGSEET] Working Group on Software Engineering Education and Training (WGSEET). Available WWW <URL: http://www.sei.cmu.edu/topics/collaborating/ed/workgroup-ed.html> (1999).

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (LEAVE BLANK)

2. REPORT DATE
November 1999

3. REPORT TYPE AND DATES COVERED
Final

4. TITLE AND SUBTITLE
Guidelines for Software Engineering Education, Version 1

5. FUNDING NUMBERS
C — F19628-95-C-0003

6. AUTHOR(S)
Donald Bagert, Thomas Hilburn, Greg Hislop, Michael Lutz, Michael McCracken, Susan Mengel

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

7. PERFORMING ORGANIZATION REPORT NUMBER
CMU/SEI-99-TR-032

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
HQ ESC/DIB
5 Eglin Street
Hanscom AFB, MA 01731-2116

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
ESC-TR-99-002

11. SUPPLEMENTARY NOTES

12.A DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified/Unlimited, DTIC, NTIS

12.B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

The two central parts of the *Guidelines* are the description of a software engineering body of knowledge and a curriculum model. The body of knowledge presents a high-level organization and description of the software engineering that supports effective curriculum design; and the curriculum model consists of a design architecture, a set of design concepts, and curriculum content guidance. We believe that this material and other guidance offered will provide assistance to faculty in the design and development of quality programs in software engineering and related curricula.

14. SUBJECT TERMS

14. NUMBER OF PAGES
73

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. Z39-18 298-102